Project Report

Introduction:

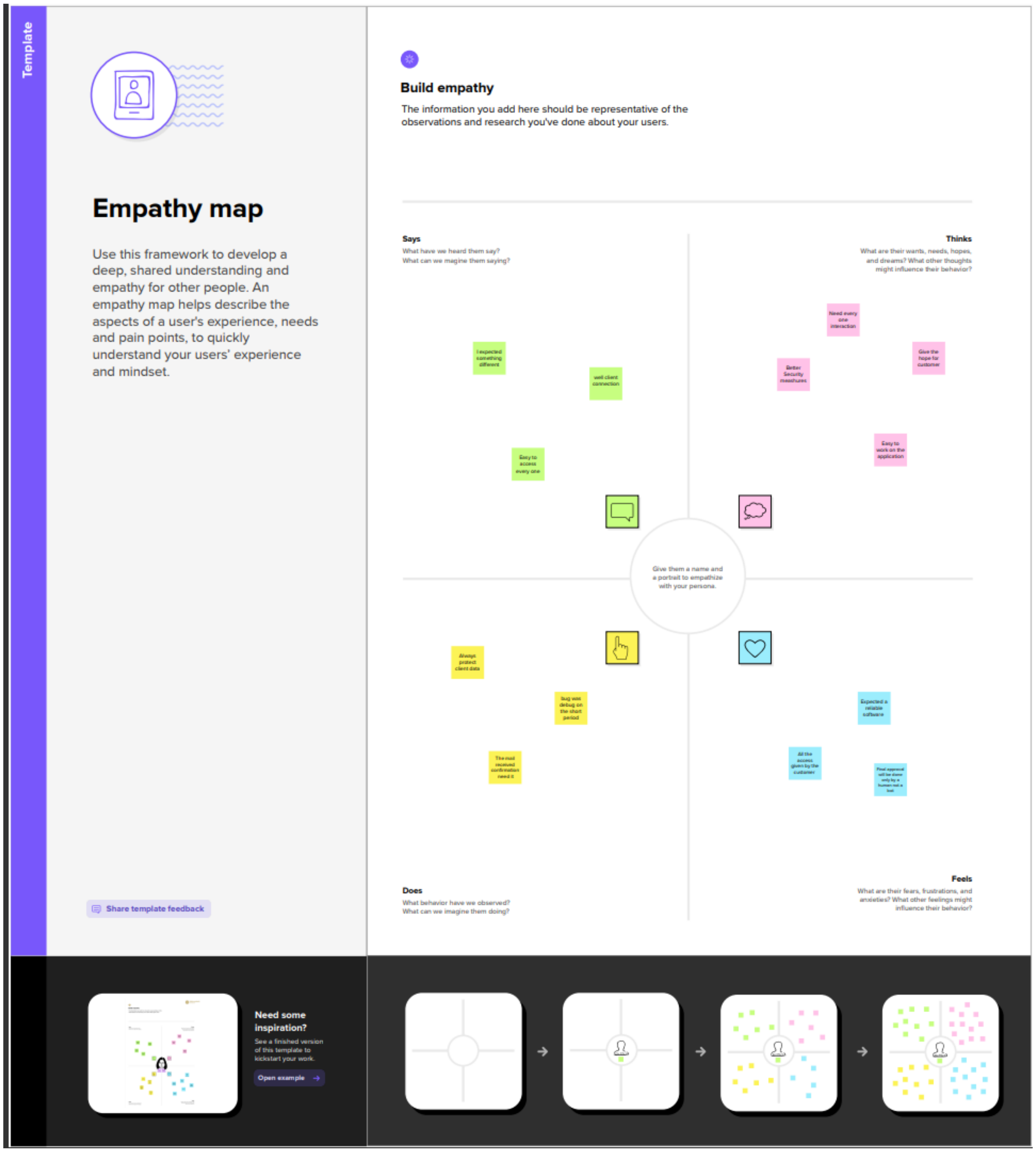
Adaptive mail is a way to **send messages** across the Internet. It's similar to traditional mail, but it also has some key differences. To get a better idea of what email is all about, take a look at the infographic below and consider how you might benefit from its use.

Purpose:

Adaptive mail is a valuable tool, it creates some challenges for writers. Miscommunication can easily occur when people have different expectations about the messages that they send and receive. Email is used for many different purposes, including contacting friends, communicating with professors and supervisors, requesting information, and applying for jobs, internships, and scholarships. Depending on your purposes, the messages you send will differ in their formality, intended audience, and desired outcomes.

Problem Definition and design thinking:

Empathy map:



Template

Empathy map

Use this framework to develop a deep, shared understanding and empathy for other people. An empathy map helps describe the aspects of a user's experience, needs and pain points, to quickly understand your users' experience and mindset.

Share template feedback

Build empathy

The information you add here should be representative of the observations and research you've done about your users.

Says
What have we heard them say?
What can we magine them saying?

I expected something different

well client connection

Easy to access every one

Thinks
What are their wants, needs, hopes, and dreams? What other thoughts might influence their behavior?

Need every one interaction

Better Security meeshures

Give the hope for customer

Easy to work on the application

Give them a name and a portrait to empathize with your persona.

Does
What behavior have we observed?
What can we imagine them doing?

Always protect client data

bug was debug on the short period

The mail received confirmation need it

Feels
What are their fears, frustrations, and anxieties? What other feelings might influence their behavior?

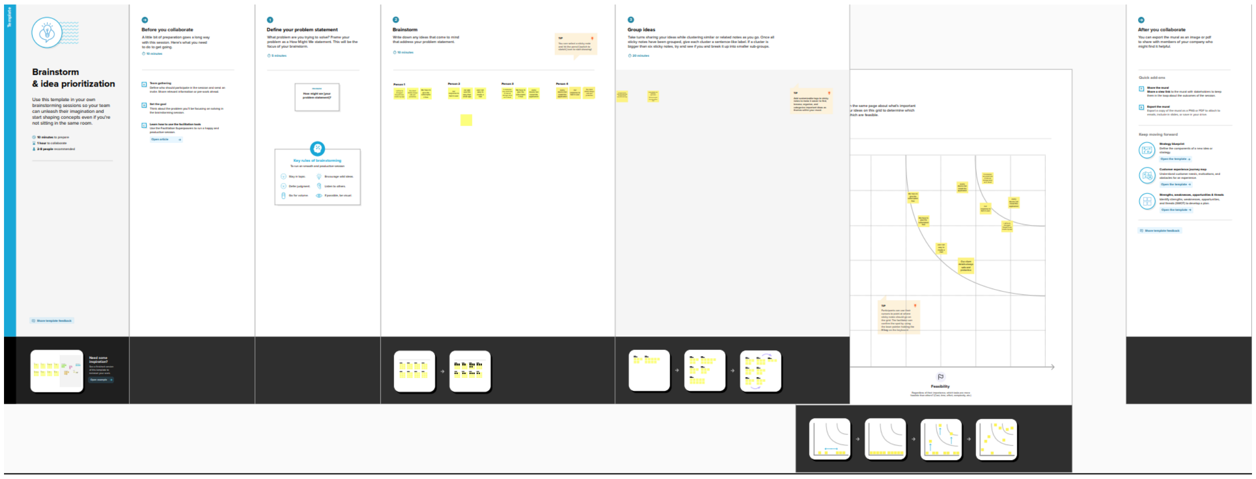
Expected a reliable software

All the access given by the customer

Final approval will be done only by a human not a bot

Need some inspiration?
See a finished version of this template to kickstart your work.
Open example

deation & Brainstorming map:



Template

Brainstorm
& idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

10 minutes to prepare
1 hour to collaborate
2-8 people recommended

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

Team gathering
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

Set the goal
Think about the problem you'll be focusing on solving in the brainstorming session.

Learn how to use the facilitation tools
Use the Facilitation Superpowers to run a happy and productive session.

Open article

1 Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

How might we [your problem statement]?

Key rules of brainstorming
To run an smooth and productive session

Stay in topic. Encourage wild ideas.
Defer judgment. Listen to others.
Go for volume. If possible, be visual.

2 Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP
You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Person 1 Person 2 Person 3 Person 4

3 Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

TIP
Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

n the same page about what's important
ur ideas on this grid to determine which
hich are feasible.

Feasibility

After you collaborate

You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

Share the mural
Share a view link to the mural with stakeholders to keep them in the loop about the outcomes of the session.

Export the mural
Export a copy of the mural as a PNG or PDF to attach to emails, include in slides, or save in your drive.

Keep moving forward

Strategy blueprint
Define the components of a new idea or strategy.
Open the template →

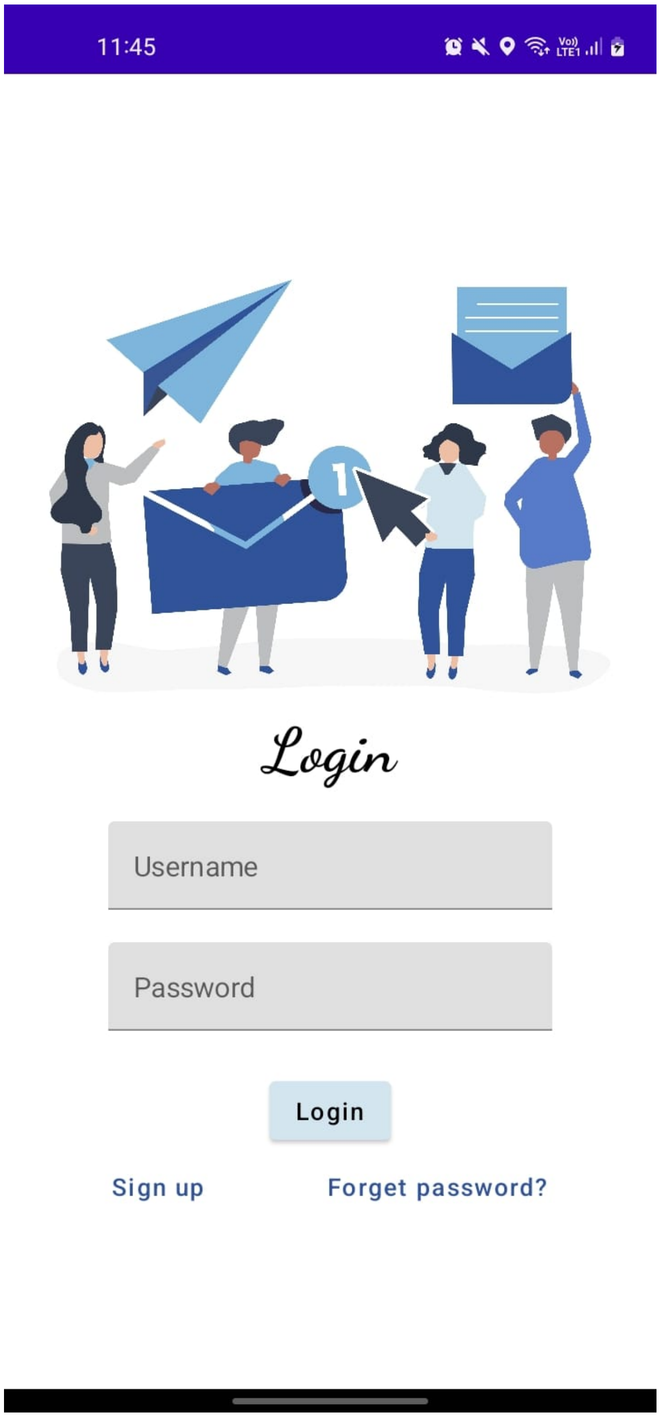
Customer experience journey map
Understand customer needs, motivations, and obstacles for an experience.
Open the template →

Strengths, weaknesses, opportunities & threats
Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.
Open the template →

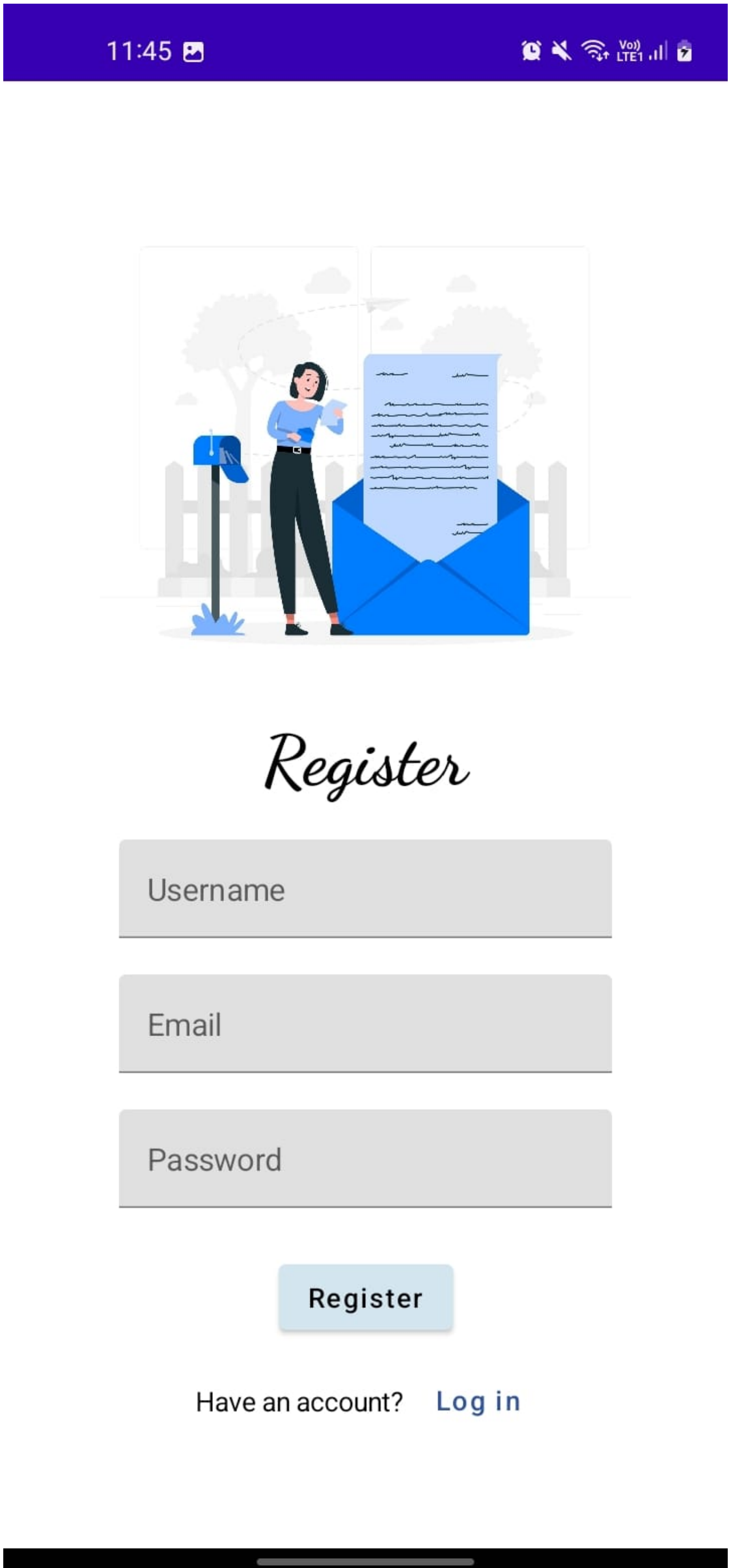
Share template feedback

Result:

Login Page:

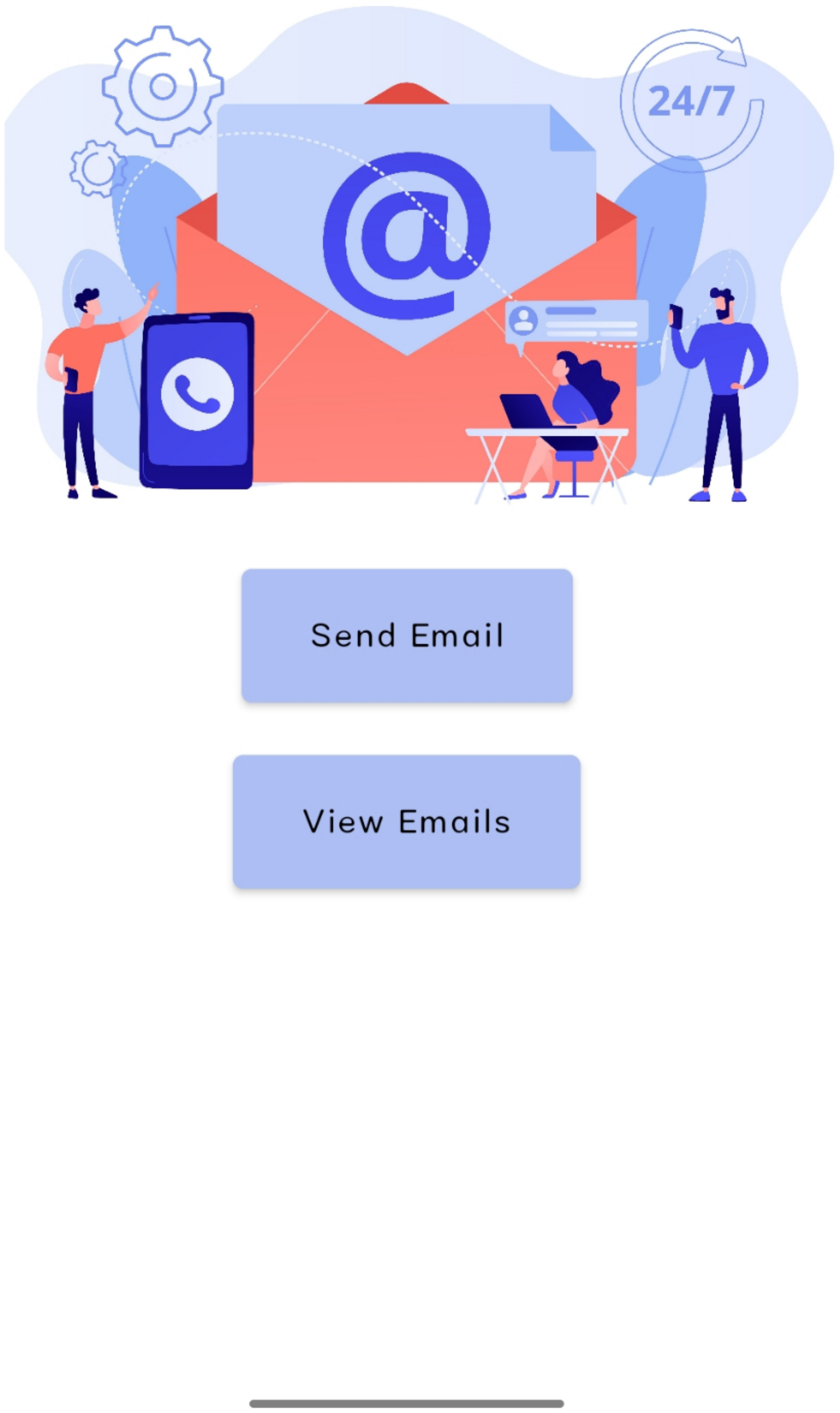


Register Image:

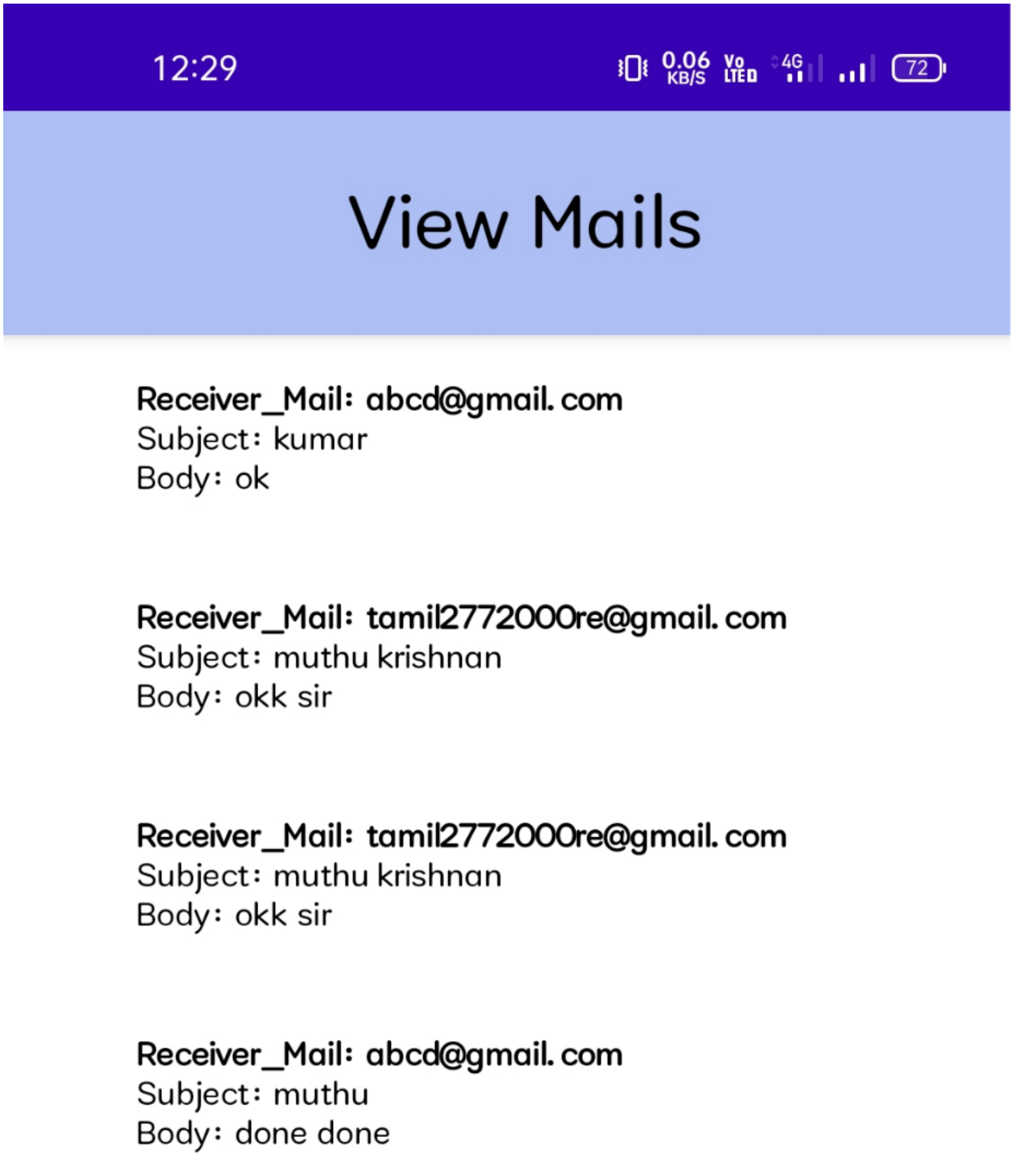

11:45
Register
Username
Email
Password
Register
Have an account?
Log in

Main Page:

Home Screen


24/7
@
Send Email
View Emails

View Mail Page:


12:29 0.06 KB/S 72

View Mails

Receiver_Mail: abcd@gmail. com
Subject: kumar
Body: ok

Receiver_Mail: tamil2772000re@gmail. com
Subject: muthu krishnan
Body: okk sir

Receiver_Mail: tamil2772000re@gmail. com
Subject: muthu krishnan
Body: okk sir

Receiver_Mail: abcd@gmail. com
Subject: muthu
Body: done done

Advantage:

- productivity tools: Adaptive mail is usually packaged with a calendar, address book, instant messaging, and more for convenience and productivity.
- Access to web services: If you want to sign up for an account like Facebook or order products from services like Amazon, you will need an email address so you can be safely identified and contacted.
- Easy mail management: Adaptive mail service providers have tools that allow you to file, label, prioritize, find, group, and filter your emails for easy management. You can even easily control spam, or junk mail.
- Privacy: Your mail is delivered to your own personal and private account with a password required to access and view emails.
- Communication with multiple people: You can send an email to multiple people at once, giving you the option to include as few as or as many people as you want in a conversation.
- Accessible anywhere at any time: You don’t have to be at home to get your mail. You can access it from any computer or mobile device that has an Internet connection.

Disadvantage:

- Adaptive Mail could potentially cause information over
- It lacks a personal touch. ...
- It can be disruptive. ...
- It cannot be ignored for a long time. ..
- It can cause misunderstandings. …
- It messages can contain viruses.

Application:

Adaptive mail is a very popular way of communicating with others over the Internet. An application that allows users to send, receive, and read email is called an *mail client*. Red Hat Enterprise Linux includes several mail applications, including graphical email clients like Evolution and Thunderbird, and text-based clients like mutt. Each of the mail client applications is designed to suit specific types of users; so, you can choose one with the features that best suits your particular needs.

The purpose of this chapter is to demonstrate how to use some of the popular email applications included in Red Hat Enterprise Linux. Since all email clients perform the same basic tasks (send mail), you should choose one that is convenient and easy to use.

Conclusion:
This project provides more creative and innovative ideas for us. To know lots of things about android and Flexible email. Its helpful for creatine bond between team member.

Future Scope:

For visual disability people we develop this app to work based on their voice message with any typing.

We provide more secure and high storage space for each message with faster transfer rate

Appendix:

Gradle scripts > build.gradle(Module :app)

```
plugins {
   id 'com.android.application'
   id 'org.jetbrains.kotlin.android'
}

android {
   namespace 'com.example.project'
   compileSdk 33

   defaultConfig {
       applicationId "com.example.project"
       minSdk 24
       targetSdk 33
       versionCode 1
       versionName "1.0"

       testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
       vectorDrawables {
           useSupportLibrary true
       }
   }

   buildTypes {
       release {
           minifyEnabled false
           proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
       }
   }
   compileOptions {
       sourceCompatibility JavaVersion.VERSION_1_8
       targetCompatibility JavaVersion.VERSION_1_8
   }
   kotlinOptions {
       jvmTarget = '1.8'
   }
   buildFeatures {
       compose true
   }
   composeOptions {
       kotlinCompilerExtensionVersion '1.2.0'
   }
   packagingOptions {
       resources {
           excludes += '/META-INF/{AL2.0,LGPL2.1}'
       }
   }
}

dependencies {

   implementation 'androidx.core:core-ktx:1.7.0'
   implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
   implementation 'androidx.activity:activity-compose:1.3.1'
   implementation "androidx.compose.ui:ui:$compose_ui_version"
   implementation "androidx.compose.ui:ui-tooling-preview:$compose_ui_version"
   implementation 'androidx.compose.material:material:1.2.0'
   testImplementation 'junit:junit:4.13.2'
   androidTestImplementation 'androidx.test.ext:junit:1.1.5'
   androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
   androidTestImplementation "androidx.compose.ui:ui-test-junit4:$compose_ui_version"
   debugImplementation "androidx.compose.ui:ui-tooling:$compose_ui_version"
   debugImplementation "androidx.compose.ui:ui-test-manifest:$compose_ui_version"

   // Adding Room dependencies
   implementation 'androidx.room:room-common:2.5.0'
   implementation 'androidx.room:room-ktx:2.5.0'
}
```

Adding User File:

```
 package com.example.project

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "user_table")
data class User(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?,
    @ColumnInfo(name = "email") val email: String?,
    @ColumnInfo(name = "password") val password: String?,
  )
```

Adding UserDao File:

```
    package com.example.project

import androidx.room.*

@Dao
interface UserDao {

   @Query("SELECT * FROM user_table WHERE email = :email")
   suspend fun getUserByEmail(email: String): User?

   @Insert(onConflict = OnConflictStrategy.REPLACE)
   suspend fun insertUser(user: User)

   @Update
   suspend fun updateUser(user: User)

   @Delete
   suspend fun deleteUser(user: User)
}
```

Adding UserDatabase:

```
package com.example.project

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [User::class], version = 1)
abstract class UserDatabase : RoomDatabase() {

   abstract fun userDao(): UserDao

   companion object {

       @Volatile
       private var instance: UserDatabase? = null

       fun getDatabase(context: Context): UserDatabase {
        return instance ?: synchronized(this) {
            val newInstance = Room.databaseBuilder(
                context.applicationContext,
                UserDatabase::class.java,
                "user_database"
            ).build()
            instance = newInstance
            newInstance
        }
    }
  }
}
```

Adding UserDatabaseHelper:

```
package com.example.project

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper


class UserDatabaseHelper(context: Context) :
  SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

  companion object {
    private const val DATABASE_VERSION = 1
    private const val DATABASE_NAME = "UserDatabase.db"

    private const val TABLE_NAME = "user_table"
    private const val COLUMN_ID = "id"
    private const val COLUMN_FIRST_NAME = "first_name"
    private const val COLUMN_LAST_NAME = "last_name"
    private const val COLUMN_EMAIL = "email"
    private const val COLUMN_PASSWORD = "password"
  }

  override fun onCreate(db: SQLiteDatabase?) {
    val createTable = "CREATE TABLE $TABLE_NAME (" +
            "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "$COLUMN_FIRST_NAME TEXT, " +
            "$COLUMN_LAST_NAME TEXT, " +
            "$COLUMN_EMAIL TEXT, " +
            "$COLUMN_PASSWORD TEXT" +
            ")"

        db?.execSQL(createTable)
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }

    fun insertUser(user: User) {
        val db = writableDatabase
        val values = ContentValues()
        values.put(COLUMN_FIRST_NAME, user.firstName)
        values.put(COLUMN_LAST_NAME, user.lastName)
        values.put(COLUMN_EMAIL, user.email)
        values.put(COLUMN_PASSWORD, user.password)
        db.insert(TABLE_NAME, null, values)
        db.close()
    }

    @SuppressLint("Range")
    fun getUserByUsername(username: String): User? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE $COLUMN_FIRST_NAME
= ?", arrayOf(username))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName = cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
        }
        cursor.close()
        db.close()
        return user
    }
    @SuppressLint("Range")
    fun getUserById(id: Int): User? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE $COLUMN_ID = ?",
arrayOf(id.toString()))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName = cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
        }
        cursor.close()
        db.close()
        return user
    }

    @SuppressLint("Range")
    fun getAllUsers(): List<User> {
        val users = mutableListOf<User>()
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
        if (cursor.moveToFirst()) {
            do {
                val user = User(
                    id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                    lastName = cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                    email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                    password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
                )
                users.add(user)
            } while (cursor.moveToNext())
        }
        cursor.close()
        db.close()
        return users
    }

}
```

Adding Email data class:

```
package com.example.project

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "email_table")
data class Email(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "receiver_mail") val recevierMail: String?,
    @ColumnInfo(name = "subject") val subject: String?,
    @ColumnInfo(name = "body") val body: String?,
)
```

Adding EmailDao interface:

```
package com.example.project

import androidx.room.*

@Dao
interface EmailDao {

  @Query("SELECT * FROM email_table WHERE  subject= :subject")
  suspend fun getOrderBySubject(subject: String): Email?

  @Insert(onConflict = OnConflictStrategy.REPLACE)
  suspend fun insertEmail(email: Email)

  @Update
  suspend fun updateEmail(email: Email)

  @Delete
  suspend fun deleteEmail(email: Email)
}
```

Adding EmailDatabase class:

```
package com.example.project

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [Email::class], version = 1)
abstract class EmailDatabase : RoomDatabase() {

  abstract fun emailDao(): EmailDao

  companion object {

     @Volatile
     private var instance: EmailDatabase? = null
```

```
      fun getDatabase(context: Context): EmailDatabase {
          return instance ?: synchronized(this) {
              val newInstance = Room.databaseBuilder(
                  context.applicationContext,
                  EmailDatabase::class.java,
                  "email_database"
              ).build()
              instance = newInstance
              newInstance
          }
      }
  }
}
```

Adding EmailDatabaseHelper class:

```
package com.example.project

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper


class EmailDatabaseHelper(context: Context) :
  SQLiteOpenHelper(context, DATABASE_NAME, null,DATABASE_VERSION){

  companion object {
      private const val DATABASE_VERSION = 1
      private const val DATABASE_NAME = "EmailDatabase.db"

      private const val TABLE_NAME = "email_table"
      private const val COLUMN_ID = "id"
      private const val COLUMN_RECEIVER_MAIL = "receiver_mail"
      private const val COLUMN_SUBJECT = "subject"
      private const val COLUMN_BODY = "body"
  }

  override fun onCreate(db: SQLiteDatabase?) {
      val createTable = "CREATE TABLE $TABLE_NAME (" +
              "${COLUMN_ID} INTEGER PRIMARY KEY AUTOINCREMENT, " +
              "${COLUMN_RECEIVER_MAIL} Text, " +
              "${COLUMN_SUBJECT} TEXT ," +
              "${COLUMN_BODY} TEXT " +
              ")"
```

```
    db?.execSQL(createTable)
  }

  override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
    db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
    onCreate(db)
  }

  fun insertEmail(email: Email) {
    val db = writableDatabase
    val values = ContentValues()
    values.put(COLUMN_RECEIVER_MAIL, email.recevierMail)
    values.put(COLUMN_SUBJECT, email.subject)
    values.put(COLUMN_BODY, email.body)
    db.insert(TABLE_NAME, null, values)
    db.close()
  }



  @SuppressLint("Range")
  fun getEmailBySubject(subject: String): Email? {
    val db = readableDatabase
    val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE $COLUMN_SUBJECT = ?",
arrayOf(subject))
    var email: Email? = null
    if (cursor.moveToFirst()) {
      email = Email(
        id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
        recevierMail = cursor.getString(cursor.getColumnIndex(COLUMN_RECEIVER_MAIL)),
        subject = cursor.getString(cursor.getColumnIndex(COLUMN_SUBJECT)),
        body = cursor.getString(cursor.getColumnIndex(COLUMN_BODY)),
      )
    }
    cursor.close()
    db.close()
    return email
  }
  @SuppressLint("Range")
  fun getEmailById(id: Int): Email? {
    val db = readableDatabase
    val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE $COLUMN_ID = ?",
arrayOf(id.toString()))
    var email: Email? = null
    if (cursor.moveToFirst()) {
      email = Email(
        id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
        recevierMail = cursor.getString(cursor.getColumnIndex(COLUMN_RECEIVER_MAIL)),
                subject = cursor.getString(cursor.getColumnIndex(COLUMN_SUBJECT)),
                body = cursor.getString(cursor.getColumnIndex(COLUMN_BODY)),
            )
        }
        cursor.close()
        db.close()
        return email
    }

    @SuppressLint("Range")
    fun getAllEmails(): List<Email> {
        val emails = mutableListOf<Email>()
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
        if (cursor.moveToFirst()) {
            do {
                val email = Email(
                    id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    recevierMail = cursor.getString(cursor.getColumnIndex(COLUMN_RECEIVER_MAIL)),
                    subject = cursor.getString(cursor.getColumnIndex(COLUMN_SUBJECT)),
                    body = cursor.getString(cursor.getColumnIndex(COLUMN_BODY)),
                )
                emails.add(email)
            } while (cursor.moveToNext())
        }
        cursor.close()
        db.close()
        return emails
    }

}
```

Adding LoginActivity.kt with database:

```
package com.example.project

import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat


import com.example.project.MainActivity
import com.example.project.R
import com.example.project.UserDatabaseHelper

class LoginActivity : ComponentActivity() {
  private lateinit var databaseHelper: UserDatabaseHelper
  override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      databaseHelper = UserDatabaseHelper(this)
      setContent {

          LoginScreen(this, databaseHelper)
      }
  }
}
@Composable
fun LoginScreen(context: Context, databaseHelper: UserDatabaseHelper) {


  var username by remember { mutableStateOf("") }
  var password by remember { mutableStateOf("") }
  var error by remember { mutableStateOf("") }

  Column(
      modifier = Modifier.fillMaxSize().background(Color.White),
      horizontalAlignment = Alignment.CenterHorizontally,
      verticalArrangement = Arrangement.Center
  ) {
      Image(
          painterResource(id = R.drawable.email_login), contentDescription = ""
      )


      Text(
          fontSize = 36.sp,
          fontWeight = FontWeight.ExtraBold,
          fontFamily = FontFamily.Cursive,
          text = "Login"
      )
      Spacer(modifier = Modifier.height(10.dp))

      TextField(
          value = username,
          onValueChange = { username = it },
          label = { Text("Username") },
          modifier = Modifier.padding(10.dp)
              .width(280.dp)
      )

      TextField(
          value = password,
          onValueChange = { password = it },
          label = { Text("Password") },
          visualTransformation = PasswordVisualTransformation(),
          modifier = Modifier.padding(10.dp)
              .width(280.dp)
      )

      if (error.isNotEmpty()) {
          Text(
              text = error,
              color = MaterialTheme.colors.error,
              modifier = Modifier.padding(vertical = 16.dp)
          )
      }

      Button(
          onClick = {
              if (username.isNotEmpty() && password.isNotEmpty()) {
                  val user = databaseHelper.getUserByUsername(username)
                  if (user != null && user.password == password) {
                      error = "Successfully log in"
                      context.startActivity(
                          Intent(
                              context,
                              MainActivity::class.java
                          )
                      )
                      //onLoginSuccess()
                  }

              } else {
                  error = "Please fill all fields"
              }
```

```
          },
          colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFd3e5ef)),
          modifier = Modifier.padding(top = 16.dp)
      ) {
          Text(text = "Login")
      }
      Row {
          TextButton(onClick = {context.startActivity(
              Intent(
                  context,
                  RegisterActivity::class.java
              )
          )}
          )
          { Text(color = Color(0xFF31539a),text = "Sign up") }
          TextButton(onClick = {
          })

          {
              Spacer(modifier = Modifier.width(60.dp))
              Text(color = Color(0xFF31539a),text = "Forget password?")
          }
      }
   }
}
private fun startMainPage(context: Context) {
   val intent = Intent(context, MainActivity::class.java)
   ContextCompat.startActivity(context, intent, null)
}
```

Adding RegisterActivity.kt with database:

```
package com.example.project

import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat


import com.example.project.R
import com.example.project.UserDatabaseHelper

class RegisterActivity : ComponentActivity() {
  private lateinit var databaseHelper: UserDatabaseHelper
  override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      databaseHelper = UserDatabaseHelper(this)
      setContent {

          RegistrationScreen(this, databaseHelper)
      }
  }
}

@Composable
fun RegistrationScreen(context: Context, databaseHelper: UserDatabaseHelper) {



  var username by remember { mutableStateOf("") }
  var password by remember { mutableStateOf("") }
  var email by remember { mutableStateOf("") }
  var error by remember { mutableStateOf("") }

  Column(
      modifier = Modifier.fillMaxSize().background(Color.White),
      horizontalAlignment = Alignment.CenterHorizontally,
      verticalArrangement = Arrangement.Center
  ) {
      Image(
          painterResource(id = R.drawable.email_signup), contentDescription = "",
          modifier = Modifier.height(300.dp)
      )
      Text(
          fontSize = 36.sp,
          fontWeight = FontWeight.ExtraBold,
```

```kotlin
            fontFamily = FontFamily.Cursive,
            text = "Register"
        )

        Spacer(modifier = Modifier.height(10.dp))
        TextField(
            value = username,
            onValueChange = { username = it },
            label = { Text("Username") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp)

        )

        TextField(
            value = email,
            onValueChange = { email = it },
            label = { Text("Email") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp)
        )

        TextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("Password") },
            visualTransformation = PasswordVisualTransformation(),
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp)
        )


        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = MaterialTheme.colors.error,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }

        Button(
            onClick = {
                if (username.isNotEmpty() && password.isNotEmpty() && email.isNotEmpty()) {
                    val user = User(
                        id = null,
                        firstName = username,
                        lastName = null,
                        email = email,
                        password = password
                    )
                    databaseHelper.insertUser(user)
                    error = "User registered successfully"
                    // Start LoginActivity using the current context
                    context.startActivity(
                        Intent(
                            context,
                            LoginActivity::class.java
                        )
                    )

                } else {
                    error = "Please fill all fields"
                }
            },
            colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFd3e5ef)),
            modifier = Modifier.padding(top = 16.dp)
        ) {
            Text(text = "Register")
        }
        Spacer(modifier = Modifier.width(10.dp))
        Spacer(modifier = Modifier.height(10.dp))

        Row() {
            Text(
                modifier = Modifier.padding(top = 14.dp), text = "Have an account?"
            )
            TextButton(onClick = {
                context.startActivity(
                    Intent(
                        context,
                        LoginActivity::class.java
                    )
                )
            })

            {
                Spacer(modifier = Modifier.width(10.dp))
                Text(color = Color(0xFF31539a),text = "Log in")
            }
        }
    }
}
private fun startLoginActivity(context: Context) {
    val intent = Intent(context, LoginActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
```

```
}
```

Adding MainActivity.kt file:

```
package com.example.project

import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat
import androidx.core.content.ContextCompat.startActivity



class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContent {
           // A surface container using the 'background' color from the theme
           Surface(
               modifier = Modifier.fillMaxSize().background(Color.White),
           ) {
               Email(this)
           }

       }
   }
}

@Composable
```

```
fun Email(context: Context) {
  Text(
    text = "Home Screen",
    modifier = Modifier.padding(top = 74.dp, start = 100.dp, bottom = 24.dp),
    color = Color.Black,
    fontWeight = FontWeight.Bold,
    fontSize = 32.sp
  )

  Column(
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Center
  ) {
    Image(
      painterResource(id = R.drawable.home_screen), contentDescription = ""
    )


    Button(onClick = {
      context.startActivity(
        Intent(
          context,
          SendMailActivity::class.java
        )
      )
    },
      colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFadbef4))
    ) {
      Text(
        text = "Send Email",
        modifier = Modifier.padding(10.dp),
        color = Color.Black,
        fontSize = 15.sp
      )
    }

    Spacer(modifier = Modifier.height(20.dp))

    Button(onClick = {
      context.startActivity(
        Intent(
          context,
          ViewMailActivity::class.java
        )
      )
    },
      colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFadbef4))
    ) {
      Text(
            text = "View Emails",
            modifier = Modifier.padding(10.dp),
            color = Color.Black,
            fontSize = 15.sp
        )
    }


  }
}
```

Adding SendMailActivity.kt file:

```
package com.example.project

import android.annotation.SuppressLint
import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.project.Email

class SendMailActivity : ComponentActivity() {
  private lateinit var databaseHelper: EmailDatabaseHelper
  @SuppressLint("UnusedMaterialScaffoldPaddingParameter")
  override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      databaseHelper = EmailDatabaseHelper(this)
      setContent {

          Scaffold(
            // in scaffold we are specifying top bar.
```

```
                topBar = {
                    // inside top bar we are specifying
                    // background color.
                    TopAppBar(backgroundColor = Color(0xFFadbef4), modifier = Modifier.height(80.dp),
                        // along with that we are specifying
                        // title for our top bar.
                        title = {
                            // in the top bar we are specifying
                            // title as a text
                            Text(
                                // on below line we are specifying
                                // text to display in top app bar.
                                text = "Send Mail",
                                fontSize = 32.sp,
                                color = Color.Black,

                                // on below line we are specifying
                                // modifier to fill max width.
                                modifier = Modifier.fillMaxWidth(),

                                // on below line we are
                                // specifying text alignment.
                                textAlign = TextAlign.Center,
                            )
                        }
                    )
                }
            ) {
                // on below line we are
                // calling method to display UI.
                openEmailer(this,databaseHelper)
            }
        }
    }
}
@Composable
fun openEmailer(context: Context, databaseHelper: EmailDatabaseHelper)  {

   // in the below line, we are
   // creating variables for URL
   var recevierMail by remember {mutableStateOf("") }
   var subject by remember {mutableStateOf("") }
   var body by remember {mutableStateOf("") }
   var error by remember { mutableStateOf("") }

   // on below line we are creating
   // a variable for a context
   val ctx = LocalContext.current
```

```
  // on below line we are creating a column
  Column(
    // on below line we are specifying modifier
    // and setting max height and max width
    // for our column
    modifier = Modifier
        .fillMaxSize()
        .padding(top = 55.dp, bottom = 25.dp, start = 25.dp, end = 25.dp),
    horizontalAlignment = Alignment.Start
  ) {

    // on the below line, we are
    // creating a text field.
    Text(text = "Receiver Email-Id",
        fontWeight = FontWeight.Bold,
        fontSize = 16.sp)
    TextField(
        // on below line we are specifying
        // value for our  text field.
        value = receviermail,

        // on below line we are adding on value
        // change for text field.
        onValueChange = { receviermail = it },

        // on below line we are adding place holder as text
        label = { Text(text = "Email address") },
        placeholder = { Text(text = "abc@gmail.com") },

        // on below line we are adding modifier to it
        // and adding padding to it and filling max width
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth(),

        // on below line we are adding text style
        // specifying color and font size to it.
        textStyle = TextStyle(color = Color.Black, fontSize = 15.sp),

        // on below line we are
        // adding single line to it.
        singleLine = true,
    )
    // on below line adding a spacer.
    Spacer(modifier = Modifier.height(10.dp))

    Text(text = "Mail Subject",
        fontWeight = FontWeight.Bold,
        fontSize = 16.sp)
    // on the below line, we are creating a text field.
    TextField(
        // on below line we are specifying
        // value for our  text field.
        value = subject,

        // on below line we are adding on value change
        // for text field.
        onValueChange = { subject = it },

        // on below line we are adding place holder as text
        placeholder = { Text(text = "Subject") },

        // on below line we are adding modifier to it
        // and adding padding to it and filling max width
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth(),

        // on below line we are adding text style
        // specifying color and font size to it.
        textStyle = TextStyle(color = Color.Black, fontSize = 15.sp),

        // on below line we are
        // adding single line to it.
        singleLine = true,
    )

    // on below line adding a spacer.
    Spacer(modifier = Modifier.height(10.dp))

    Text(text = "Mail Body",
        fontWeight = FontWeight.Bold,
        fontSize = 16.sp)
    // on the below line, we are creating a text field.
    TextField(
        // on below line we are specifying
        // value for our  text field.
        value = body,

        // on below line we are adding on value
        // change for text field.
        onValueChange = { body = it },

        // on below line we are adding place holder as text
        placeholder = { Text(text = "Body") },

        // on below line we are adding modifier to it
        // and adding padding to it and filling max width
```

```
            modifier = Modifier
                .padding(16.dp)
                .fillMaxWidth(),

            // on below line we are adding text style
            // specifying color and font size to it.
            textStyle = TextStyle(color = Color.Black, fontSize = 15.sp),

            // on below line we are
            // adding single line to it.
            singleLine = true,
        )

        // on below line adding a spacer.
        Spacer(modifier = Modifier.height(20.dp))

        // on below line adding a
        // button to send an email
        Button(onClick = {

            if( recevierMail.isNotEmpty() && subject.isNotEmpty() && body.isNotEmpty()) {
                val email = Email(id = null, recevierMail = recevierMail, subject = subject, body = body)
                databaseHelper.insertEmail(email)
                error = "Mail Saved"
            } else {
                error = "Please fill all fields"
            }

            // on below line we are creating
            // an intent to send an email
            val i = Intent(Intent.ACTION_SEND)

            // on below line we are passing email address,
            // email subject and email body
            val emailAddress = arrayOf(recevierMail)
            i.putExtra(Intent.EXTRA_EMAIL,emailAddress)
            i.putExtra(Intent.EXTRA_SUBJECT,subject)
            i.putExtra(Intent.EXTRA_TEXT,body)

            // on below line we are
            // setting type of intent
            i.setType("message/rfc822")

            // on the below line we are starting our activity to open email application.
            ctx.startActivity(Intent.createChooser(i,"Choose an Email client : "))

        },
            colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFd3e5ef))
        ) {
            // on the below line creating a text for our button.
            Text(
                // on below line adding a text ,
                // padding, color and font size.
                text = "Send Email",
                modifier = Modifier.padding(10.dp),
                color = Color.Black,
                fontSize = 15.sp
            )
        }
    }
}
```

Adding ViewMailActivity.kt file:

```
package com.example.project

import android.annotation.SuppressLint
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.layout.R
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.LazyRow
import androidx.compose.foundation.lazy.items
import androidx.compose.material.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

import com.example.project.EmailDatabaseHelper

class ViewMailActivity : ComponentActivity() {
  private lateinit var emailDatabaseHelper: EmailDatabaseHelper
  @SuppressLint("UnusedMaterialScaffoldPaddingParameter")
  override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
```

```
    emailDatabaseHelper = EmailDatabaseHelper(this)
    setContent {

        Scaffold(
            // in scaffold we are specifying top bar.
            topBar = {
                // inside top bar we are specifying
                // background color.
                TopAppBar(backgroundColor = Color(0xFFadbef4), modifier = Modifier.height(80.dp),
                    // along with that we are specifying
                    // title for our top bar.
                    title = {
                        // in the top bar we are specifying
                        // title as a text
                        Text(
                            // on below line we are specifying
                            // text to display in top app bar.
                            text = "View Mails",
                            fontSize = 32.sp,
                            color = Color.Black,

                            // on below line we are specifying
                            // modifier to fill max width.
                            modifier = Modifier.fillMaxWidth(),

                            // on below line we are
                            // specifying text alignment.
                            textAlign = TextAlign.Center,
                        )
                    }
                )
            }
        ) {
            val data = emailDatabaseHelper.getAllEmails();
            Log.d("swathi", data.toString())
            val email = emailDatabaseHelper.getAllEmails()
            ListListScopeSample(email)
        }
    }
  }
}
@Composable
fun ListListScopeSample(email: List<Email>) {
  LazyRow(
    modifier = Modifier
        .fillMaxSize(),
    horizontalArrangement = Arrangement.SpaceBetween
  ) {
    item {
        LazyColumn {
            items(email) { email ->
                Column(
                    modifier = Modifier.padding(
                        top = 16.dp,
                        start = 48.dp,
                        bottom = 20.dp
                    )
                ) {
                    Text("Receiver_Mail: ${email.recevierMail}", fontWeight = FontWeight.Bold)
                    Text("Subject: ${email.subject}")
                    Text("Body: ${email.body}")
                }
            }
        }
    }

  }
}
```